**MP2: Cryptocurrency Report**

Zhicong Fan(zhicong2), Kehang Chang(kehang2)

04/16/2020

cluster number: g35

Git Revision Number:

Git URL: <https://gitlab.engr.illinois.edu/zhicong2/cs425mps_group35>

**Part1 Transaction propagation:**

**Introduction:**

In this mp, we implemented a transaction propagation scheme via gossip method, and then we used Nakamoto consensus across all nodes over the network to enforce total-order balance transaction for all nodes on the network. ­­­

**Running Instruction:**

**We uploaded separate files for part1 and part2**

Part1: If you want to only test part1, please do:

go build client.go

./client <node\_name> <node\_port\_number>

Please use the same port, because we implemented in that way

Part2: If you want to test part2(this file includes both part1 and part2), please do:

go build client\_part2.go

./client\_part2 <node\_name> <node\_port\_number>

Please use the same port, because we implemented in that way

**Part 1: Transaction propagation**

1. How your nodes keep connectivity; how they discover nodes beyond the originally introduced ones, and how they detect failed nodes. You should justify why you think your design is robust to failures.

In our design, when one node enters the network, it will get introduced to no more than three neighboring nodes by mp2\_service.py. Then these neighboring nodes will dial back to the original nodes if they don’t have any descendants or their descendants will dial back if they have any. Based on our design, the connected network will be a fully connected bidirectional graph to ensure gossip message propagation.

Each node in the graph is restricted to have no more than 14 outbound nodes and 128 inbound nodes. Once there is an outbound nodes failure, push subroutine will randomly request a neighboring nodes from either inbound or outbound group for one of their descendants. Then the original node will dial this introduced node with full transaction history exchange to keep graph stay fully connected and message synchronized.

2. How transactions are propagated. Describe the algorithm you are using, any parameters and how you arrived at them.

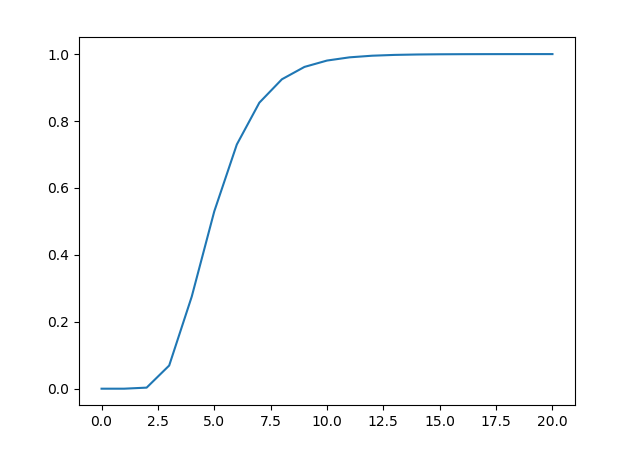
The high level of our algorithm is to let any nodes connected to the network stay in a fully connected manner so that gossip method will eventually deliver any messages to any nodes. The algorithm works as follow:

1. Each node enters the network by first dialing intro\_server (server running mp2\_service.py), then a bidirectional channel is formed.
2. The new node starts to listen on all ports and accept any requests (any new inbound nodes channels are being put into an inbound list)
3. Neighboring nodes are being introduced to the new node from intro\_server
4. The new node dials to these introduced servers (these nodes are being put into an outbound list)
5. If any introduced nodes have descendants, the descendants will dial back to the new nodes; otherwise the introduced nodes will dial back to the new nodes.
6. When two nodes are connected at the first time, their transaction history will get exchanged.
7. When there is a transaction being generated in one node, this message will be put into a transaction map and then the node will R-multicast this message to all its outbound neighbors
8. When there is a failure in the outbound list for one node, its neighbors will recommend a new node out of their descendants for this node to dial.

(In order to optimize bandwidth, propagation delay, and ensure safety, we chose the max inbound number to be 128 and outbound number to be 14)

The safety probability: ((2^k-1)/2^k)^n (k is the outbound size; n is the total number of nodes)

(safety probability vs k) (n=100) (14 is optimal pick to ensure safety)



3. What information is logged and how you used this to generate the graphs. Please make sure that the logs you used in your experiments are checked into the git repository. If you wrote any scripts to analyze the logs please include them in the repo and describe how they work.

The following information is logged:

Bandwidth/Recorded Time

Node Name/Transaction Created Time/Transaction Received Time

graph.py

Bandwidth can be used to calculated total bandwidth

Max propagation time can be calculated: Transaction Received Time - Transaction Created Time

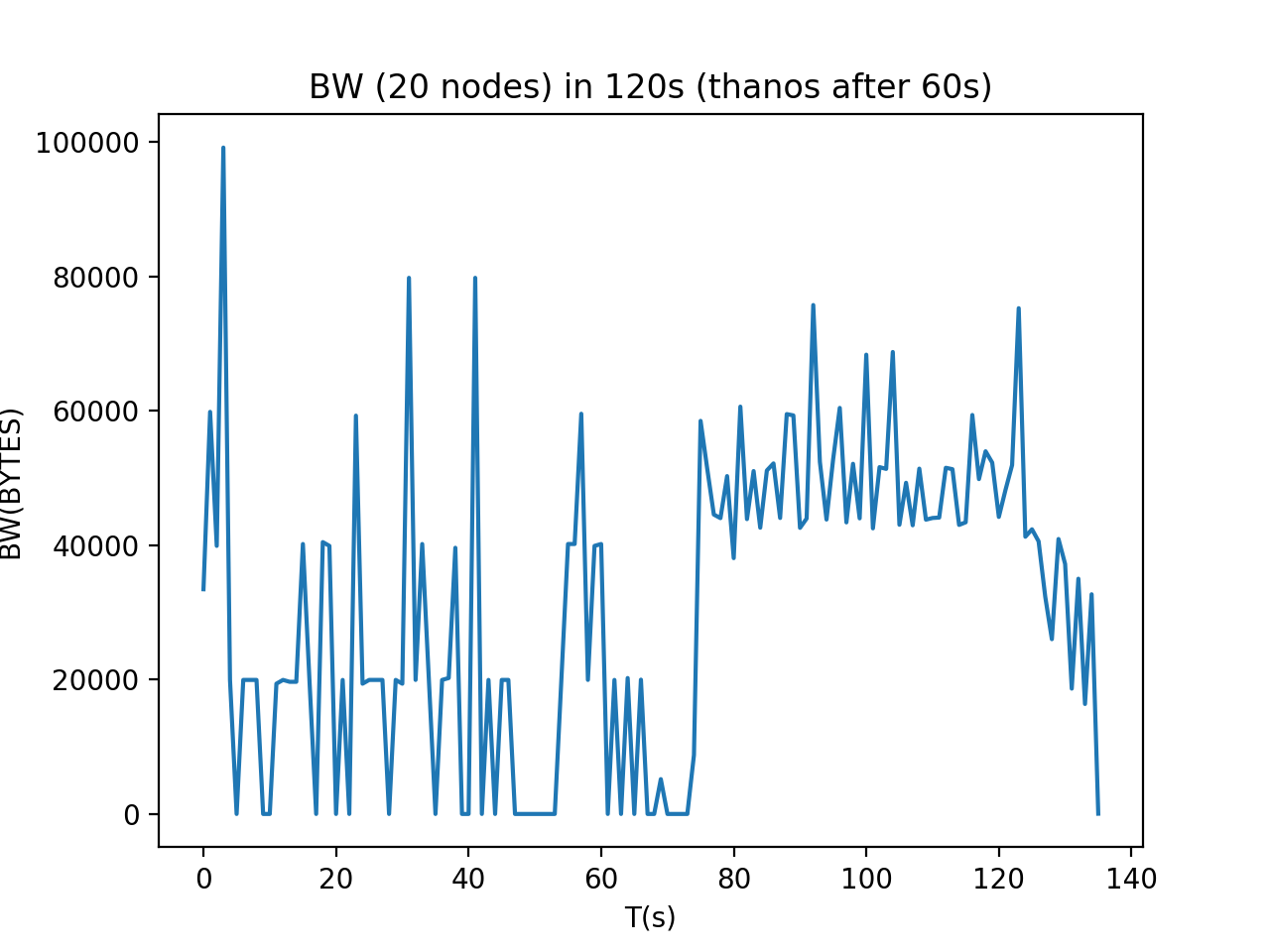
log\_service.go

The script is used to receive data from other VMs and format data into proper ways for processing.

**Graph:**

20 Nodes Scenario:

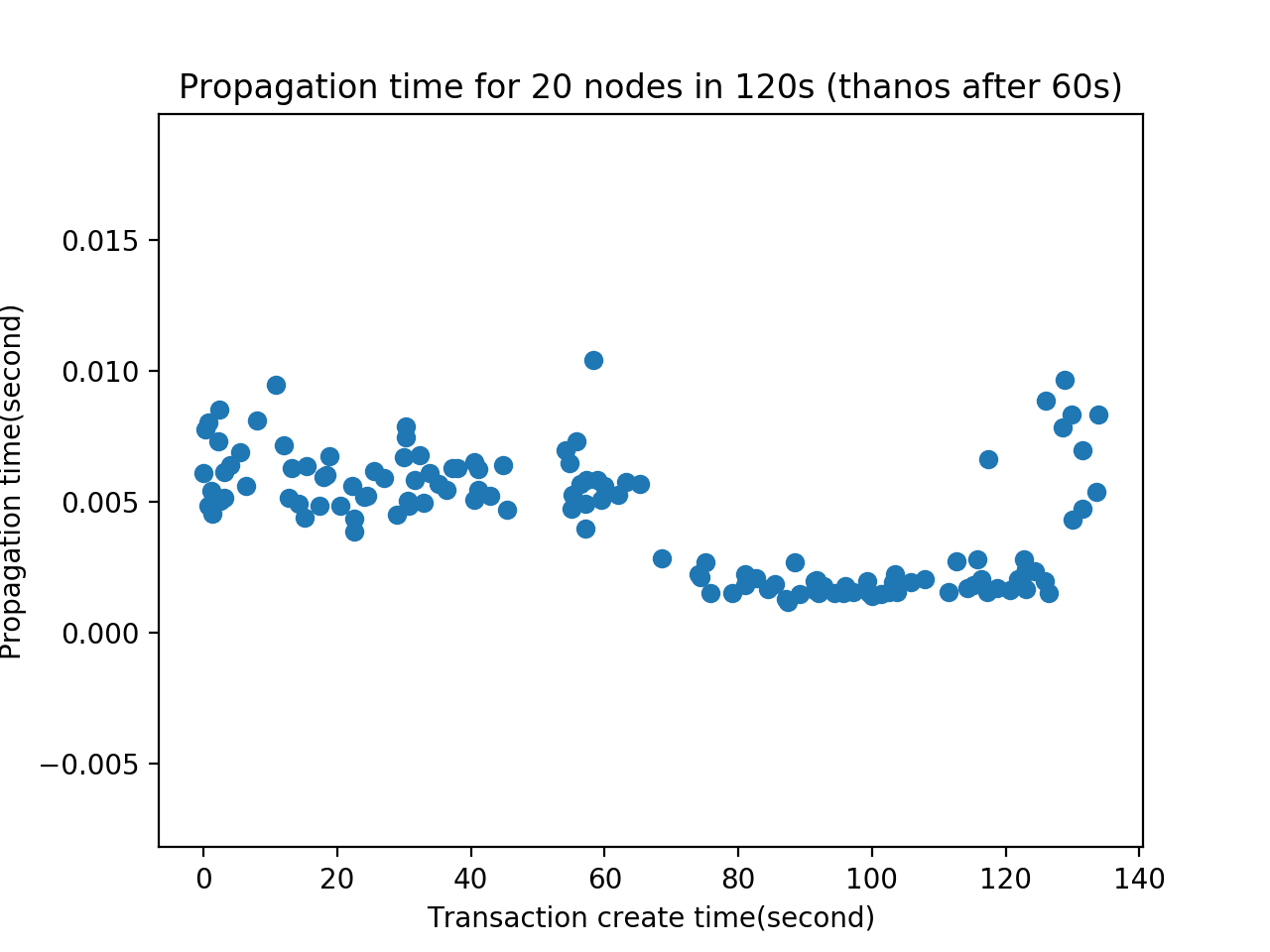
Bandwidth Graph:



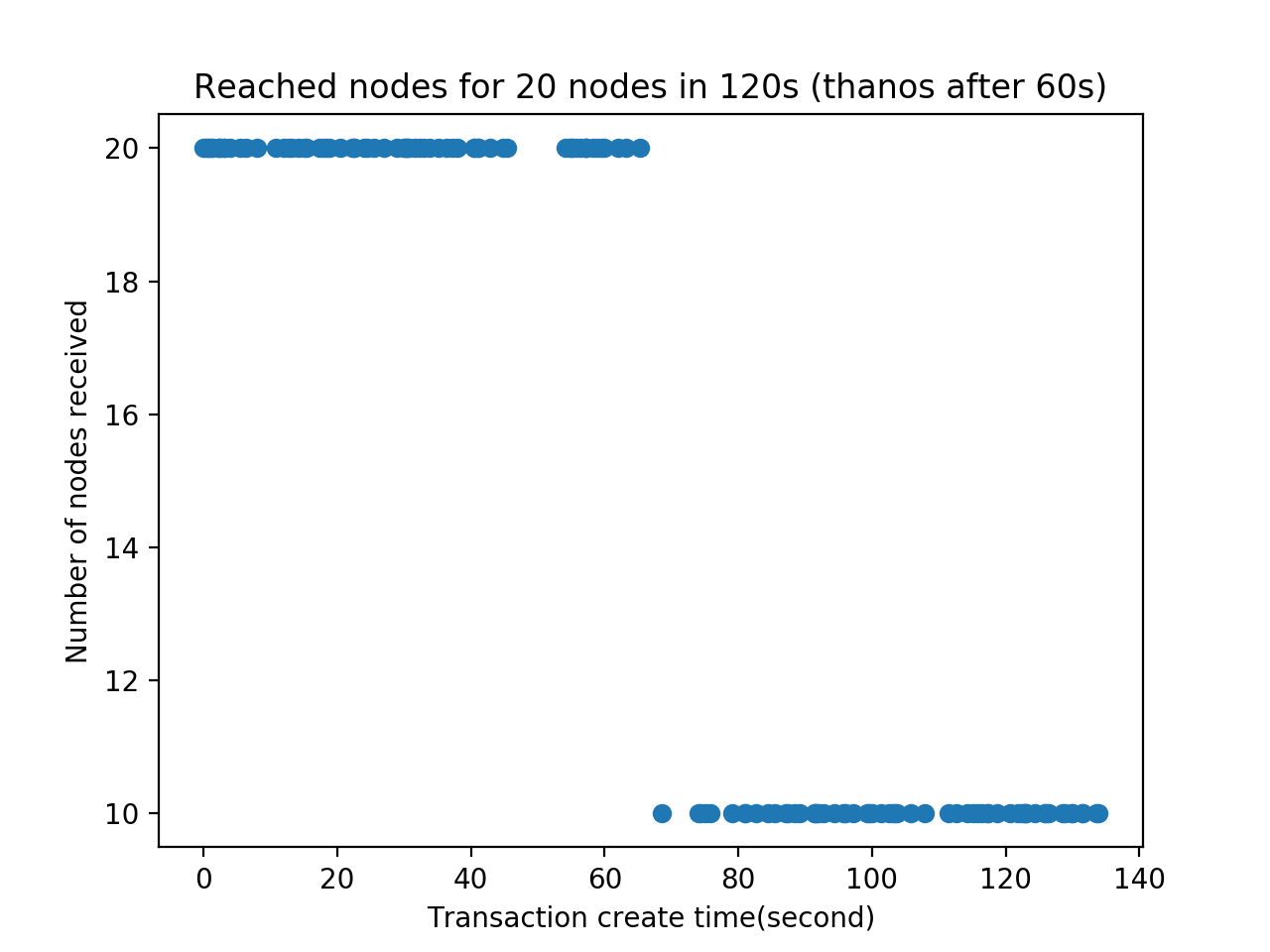
MaxPropagation Delay (for each transaction starting from time 0):

(The delay is relative high because there are 5 processes on each VM)

(Decrease number of running processes on each VM will **SIGNIFICANTLY** decrease the delay as seen in 100 nodes scenario)

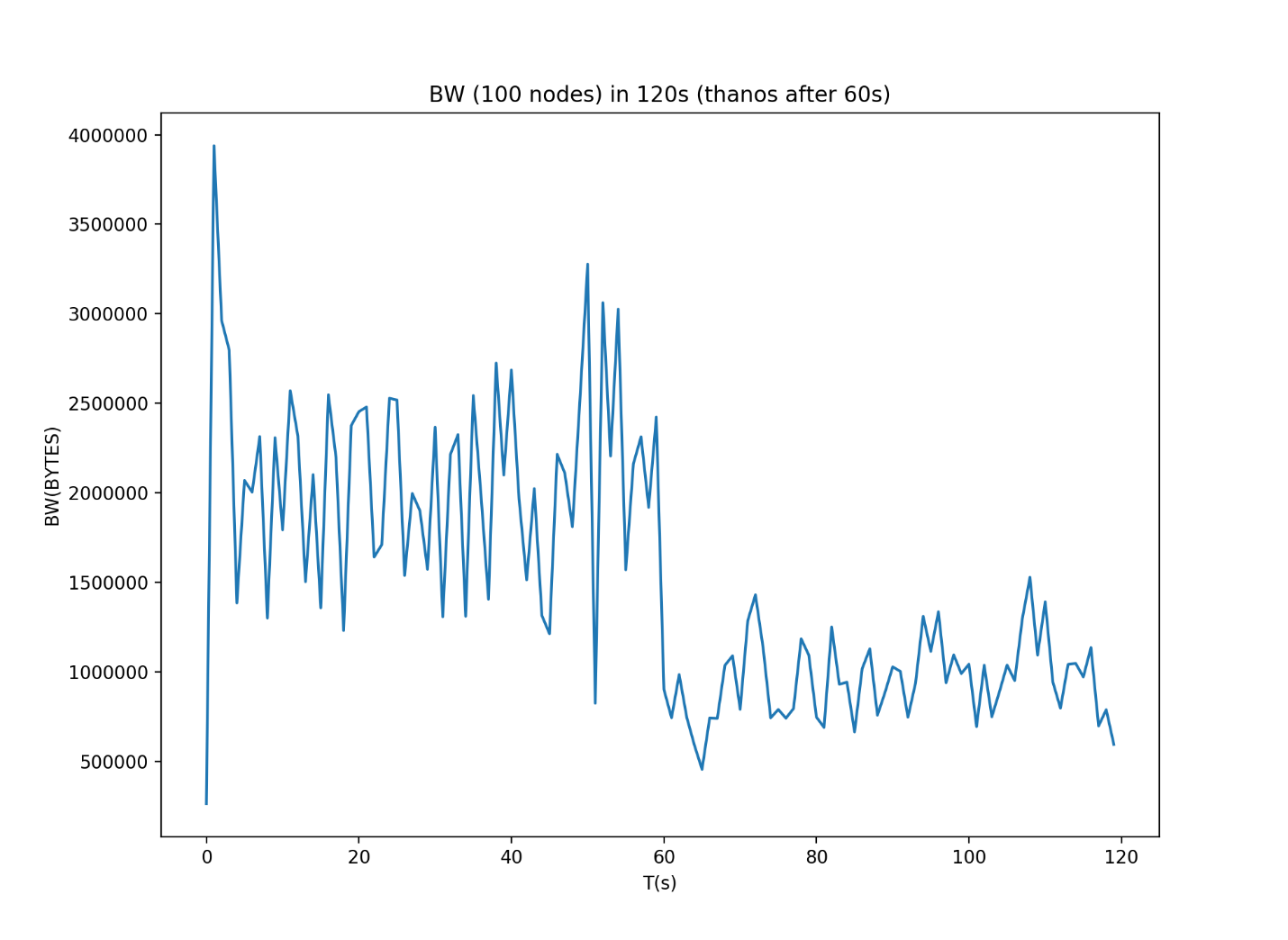
****

Number of Reached Nodes in 120sec:

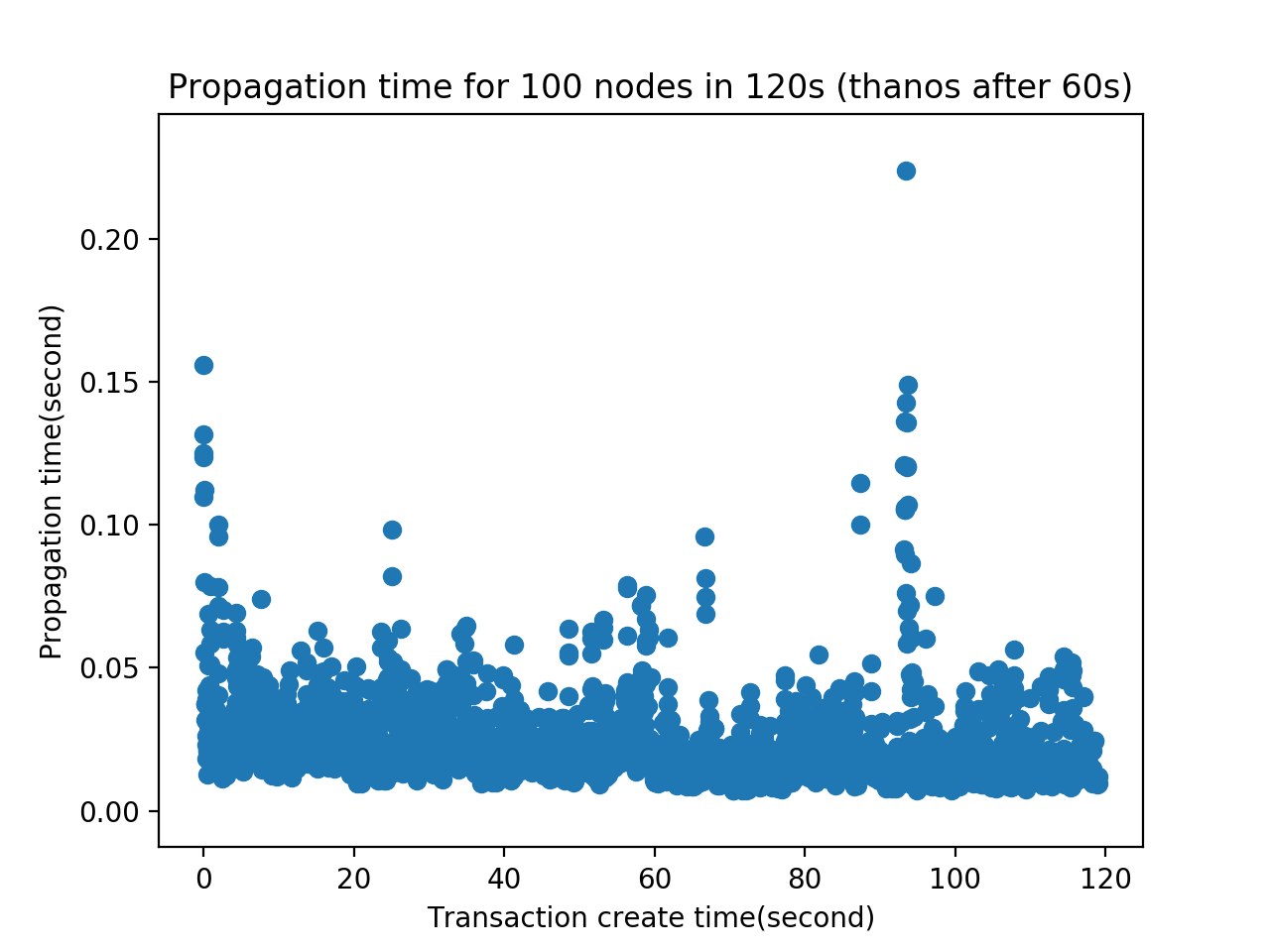
****

100 Nodes Scenario:

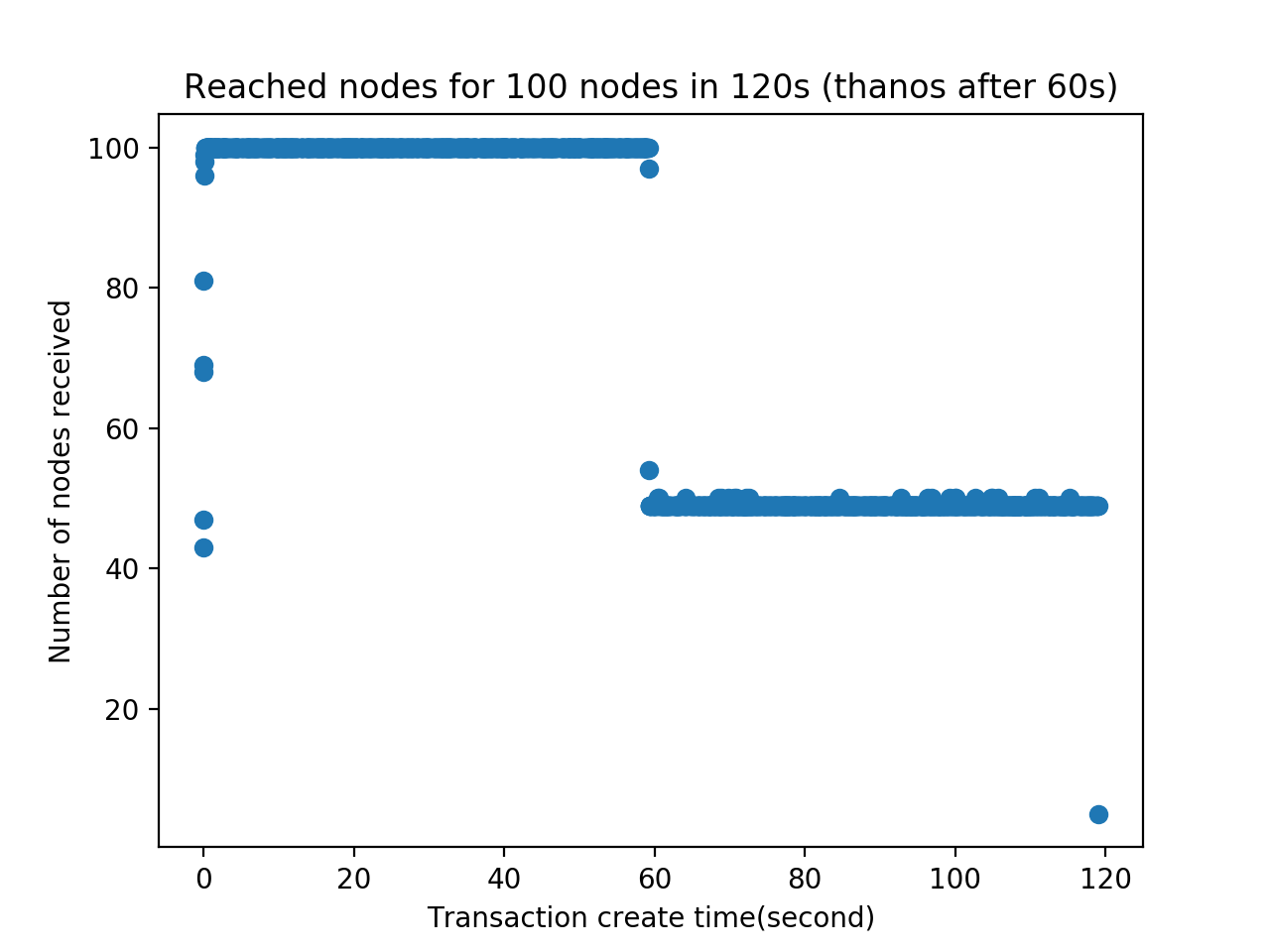
Bandwidth Graph:



MaxPropagation Delay (for each transaction starting from time 0):



Number of Reached Nodes in 120sec:

****

**Part 2: Nakamoto Consensus**

1. What the process is for validating transaction validity and collecting them into blocks.

There are 2 possible conditions when a new block is generated:

1. This block is local and has been solved

2. This block is not local, someone else sent the longer block

Then we call the “recursiveCheckAccount” function to check the balance from the very first block transactionlists. If all the transactions from the first one until the last one that we just received succeed, we then update the balance. And we remove all the used(true) transactions in our mempool.

As soon as we have updated the mempool, we generate a new block to mine. First and foremost, we lock the mempool, preventing it from being modified. And then we create a transactionlist that contains up to 2000 transactions in the mempool, set these transactions to used(true). Now we hash the whole block, and send a “SOLVE” message to the server.

2. How new blocks are propagated and imported. Discuss particularly what happens during a chain fork.

The new blocks are propagated by gossip method over the network.

There are two scenarios for new blocks to be imported:

1. If the current node already has mined a block and its length is same as the new block from the network, it will ignore the new block from the network

(Only when the new block has a longer length, the node will accept the new block)

1. If the current node hasn’t mined a block yet, it will accept based on first-come-first-serve principle if all new blocks have same length

(However, a longer length new block will have the highest priority)

A chain fork happens when there are multiple nodes that mined a block. Each chain will keep appending new blocks on their chains. All forks will eventually merge when there is one chain that is longest among all forks.

3. What information is logged and how you used this to generate the graphs. Please make sure that the logs you used in your experiments are checked into the git repository. If you wrote any scripts to analyze the logs please include them in the repo and describe how they work.

Graph:

Language used: “Go” for client, “Python” for plotting the graph